Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Programación Dinámica

18 de septiembre 2023

Tobías Serpa
108266

Carolina Mauro
108294

Valentina Adelsflugel
108201

1. Objetivo

El siguiente trabajo tiene como objetivo el desarrollo de un algoritmo que logre determinar la mejor manera de organizar los días de entrenamiento y descanso de la Scaloneta, para obtener la mayor ganancia posible. Para ello, se deben tener en cuenta las limitaciones de la energía de los jugadores a medida que entrenan j días seguidos, y las ganancias asociadas a cada día de entrenamiento. Para maximizar la ganania, es necesario realizar un profundo análisis del problema y sus variables, con el fin de generar la ecuación de recurrencia. Luego, plasmar dicha ecuación en código, respetando los principios de la programación dinámica. Hecho esto, se debe analizar la complejidad del algoritmo generado, y cómo moficaciones de las variables involucradas en el problema, generan cambios en los tiempos de ejecución y optimalidad del algoritmo.

2. ¿Qué es Programación Dinámica?

Es una técnica de programación, tal como División y Conquista y Algoritmos Greedy. Una idea básica de la programación dinámica es no hacer recálculos cuando es evitable, siendo esto posible por medio del concepto de Memoization.

Programación dinámica es una técnica de construcción iterativa de soluciones a partir de subproblemas: se basa en la capacidad de descomponer un problema original en subproblemas más pequeños, y por ende, más sencillos de analizar. A través de una lógica inductiva, aborda la pregunta clave: '¿Es posible utilizar las soluciones previamente calculadas para construir la solución que necesitamos?' Esta metodología implica una exploración implícita del espacio de posibles soluciones, aprovechando la intuición proporcionada por Memoization para identificar y utilizar los subproblemas en la construcción de soluciones más grandes. El enfoque iterativo y Memoization nos permiten evitar la exploración exhaustiva de un espacio exponencial de soluciones, lo que resulta en una significativa reducción en la complejidad temporal de nuestros algoritmos.

Cuando nos planteamos si podemos aplicar programación dinámica (PD) a un problema específico, es importante considerar ciertos criterios clave. En primer lugar, es fundamental que el problema pueda dividirse en un número polinomial de subproblemas. Además, debe ser posible construir la solución al problema original a partir de las soluciones a estos subproblemas. Así mismo, es crucial que exista un orden natural en la resolución de los subproblemas, donde los problemas 'mayores' se resuelven mediante la composición de problemas 'menores'. En este sentido, la versión recursiva de PD aborda los subproblemas en un formato Top-Down, mientras que la versión iterativa sigue un enfoque Bottom-Up, siendo esta última la preferida. Se sostiene esta preferencia por el modo iterativo, ya que es más sencillo enteder qué pasa y cuándo pasa, facilitando esto el cálculo de la complejidad algorítmica.

3. Análisis del problema

Primero, definimos las variables que aparecen en nuestro problema:

$n \rightarrow$ Cantidad de días de entrenamiento.

$e_i \rightarrow$Esfuerzo requerido el día i (equivale a la ganancia de entrenar ese día)

$s_j \rightarrow$ Energía disponible con la que se cuenta al día $1, 2, 3, ..., n$ de estar entrenando sin haber descansado previamente. Aclaración: j no necesariamente es igual a i.

$g_i \rightarrow$ Ganancia obtenida el día i en función de la energía disponible s_j. Donde: $g_i = \text{mín}(s_j, e_i)$

Consideraciones:

- No se puede reprogramar o cambiar un entrenamiento con otro. Esto nos lleva a no poder ordenarlos con ningún criterio que nos facilite la solución.
- El día de descanso no se consigue ninguna ganancia.

- Al descansar un día, la energía al día siguiente se reinicia, siempre comenzando de nuevo desde s_1 (siendo esta la más alta disponible).

A partir de este último punto notamos que nunca va a ser conveniente descansar más de 1 día seguido. Al descansar un día, no obtenemos ninguna ganancia cuantificable en nuestra ganancia total, pero sí obtenemos una "implicita": se renueva la energía a s_1. En caso de descansar k días seguidos, la energia siempre vuelve a ser s_1. Por lo tanto, si queremos asegurarnos de obtener la máxima ganancia en el día i, basta únicamente con descansar el día anterior.

Inicialmente pensamos nuestro problema como si cada día nos encontraramos únicamante con la dicotomía entre entrenar o descansar. Esto nos llevó a querer mirarlo hacia "adelante" enfocándonos en la pregunta: ¿si entreno hoy obtengo una mayor o menor ganancia que descansando y mañana entrenar con la máxima energía disponible? Es muy fácil encontrar un contra ejemplo a este pensamiento:

Esfuerzo (e_i)	**Energía** (s_i)
20	60
60	10
55	5

Es evidente que este problema no puede ser resuelto con nuestra mentalidad inicial, debido a que no hubiesemos entrenado el primer día ya que su ganancia es inferior a la del segundo día, resultando así, una solución en la que entrenabamos el segundo y tercer día, obteniendo una ganancia total de 70. Pero se puede observar a simple vista, que lo conveniente sería entrenar el primer día, descansar, y finalizar entrenando, para con esto obtener una ganancia total de 75.

Notamos entonces que mirando nuestro problema hacia "adelante"no solo nos estaba abriendo un gran abanico de escenarios sino que además no estabamos teniendo en cuenta que al día i podríamos llegar con diferentes s_j según lo que hicimos los días anteriores. Es decir, al día i los jugadores podrían haber llegar habiendo entrenado j días seguidos con j entre 0 e $i-1$.

Notamos así que nuestro problema se reducía a encontrar para cada día cómo convenía haber llegado:

1. ¿Qué ganancia obtenemos si llegamos habiendo entrenado 0 días seguidos?
2. ¿Qué ganancia obtenemos si llegamos habiendo entrenado 1 día seguido? .

   .

   .
3. ¿Qué ganancia obtenemos si llegamos habiendo entrenado $i-1$ días seguidos?

Vemos entonces que a medida que hay más días en juego, nuestro problema escala.

- Para 1 día únicamente debemos preguntarnos ¿entreno o descanso? La respuesta siempre es entreno. Identificamos nuestro caso base.
- Para 2 días podemos llegar:

  1. Entrenando el día 1
  2. Descansando el día 1

  ¿Cómo obtengo la mejor ganancia? Miramos hacia atras: sabemos cuál es la ganancia entrenando 1 día y la comparamos con haber descansado el día 1.
- Para 3 días podemos llegar:

  1. Entrenando el día 1 y descansando el día 2 (llegamos entrenando 0 días seguidos)
  2. Entrenando el día 1 y 2 (llegamos entrenando 2 días seguidos)

3. Descansando el día 1 (llegamos entrenando 1 día seguido, el día 2)

.

.

.

- Para el día n hay que fijarse cómo obtener la máxima ganancia, siendo esta la combinación de haber llegado entrendando $n-1$ o $n-2$ o ... 0 días seguidos.

Entonces, nuestro problema crece a medida que lo hace n y la ganancia de cada día depende de cómo se llegó al mismo.

Ahora, si pensamos al día i como si fuera el último, y tenemos la combinacion de entrenamientos/descansos que nos da la máxima ganancia hasta el día $i-1$ ¿Podemos aprovechar esta información para obtener la solución en el día i? Sí, siempre y cuando almacenemos para el día $i-1$ todas las combinaciones de entrenamiento/descanso posibles (para no "recalcularlas").

3.1. ¿Cómo encontrar la solución usando Programación Dinámica?

Al momento de plantear una solución a un problema utilizando programación dinámica debemos de poder identificar:

1. El/los caso/s base
2. La forma que tienen los subproblemas
3. La forma en que dichos subproblemas se componen para solucionar subproblemas más grandes

Luego de nuestro analisis logramos identificar claramente estos 3 puntos de nuestro problema, permitiendonos afirmar que la solución al mismo iba a ser posible usando Programación Dinámica:

1. Caso base:
   a) Si contamos con 0 días: no hay días para entrenar, no obtenemos ganacia.
   b) Si contamos con 1 día: entrenamos, obtenemos máxima ganancia posible.
2. La forma que tienen los subproblemas:

   Los días entrenados en función de la ganancia y la energía disponible, sabiendo que descansar un día nos renueva la energía disponible al máximo. Los subproblemas se achican a medida que contamos con menos días.
3. La forma en que dichos subproblemas se componen para solucionar subproblemas más grandes:

   Si sabemos para $n-1$ días todas las posibilidades de combinaciones entrenar/descansar sabiendo la ganancia que da cada una, podemos obtenerlas para n días. Sabemos cuál es la ganancia si a $n-1$ días llegamos habiendo entrenado j días seguidos. Entonces, para el día n tomamos esto y calculamos la ganancia de haber llegado habiendo entrenado $j+1$ días seguidos:

   $\texttt{maxima\_ganancia\_dia\_n} = \text{máx}_{0 \leq j \leq n-1}(\texttt{entrenar\_j\_días\_seguidos} + \texttt{ganancia\_n})$

   Donde ya tenemos calculada la ganacia que se obtiene de entrenar j días seguidos y le sumamos la ganancia del día actual correspondiente, según la energía con la que contamos para cada día.

3.2. Ecuación de recurrencia

Primero vamos a ejemplificar el algoritmo matricialmente. Supongamos que tenemos los siguientes datos para $n = 10$:

Esfuerzo (e_i)	**Energía** (s_i)
36	63
2	61
78	49
19	41
59	40
79	38
65	23
64	17
33	13
41	10

Podemos armar entonces una matriz que tenga en cada columna las posibles combinaciones para cada día, según con cuántos días de entrenamiento consecutivos se haya llegado. Inicializamos una matriz de $n * n$ donde el valor inicial es 0 para cada día.

Comenzamos para $n = 1$, el cual entrenamos porque es nuestro caso base. De esta forma queda $g_1 = \text{mín}(63, 36) = 36$.

1	2	3	4	5	6	7	8	9	10	dias(n) / entrenamientos consecutivos
0	0	0	0	0	0	0	0	0	0	**10**
0	0	0	0	0	0	0	0	0	0	**9**
0	0	0	0	0	0	0	0	0	0	**8**
0	0	0	0	0	0	0	0	0	0	**7**
0	0	0	0	0	0	0	0	0	0	**6**
0	0	0	0	0	0	0	0	0	0	**5**
0	0	0	0	0	0	0	0	0	0	**4**
0	0	0	0	0	0	0	0	0	0	**3**
0	0	0	0	0	0	0	0	0	0	**2**
0	0	0	0	0	0	0	0	0	0	**1**
36	0	0	0	0	0	0	0	0	0	**0**
36	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	*Máximo alcanzado para el día i*

Ahora, agrandemos el problema para $n = 2$. ¿Cómo podríamos haber llegado hasta acá? Entrenando el día 1 con ganancia 36 o no entrenando, en este caso con ganancia acumulada 0. Por lo tanto, la primer alternativa sumada a entrenar el día 2 corresponde a hacerlo 2 días seguidos, es decir, un entrenamiento consecutivo. La ganancia acumulada entonces será $g_2 = \text{mín}(2, 61) + 36 = 38$. En cambio, entrenando hoy pero habiendo descansado el día 1, tenemos 0 entrenamientos consecutivos y $g_2 = \text{mín}(2, 63) + 0 = 2$.

No almacenamos en la matriz el caso de descansar el día 2 porque estaríamos duplicando información que ya tenemos. Se reduce a mirar el máximo obtenido una columna hacia atrás (en este caso 36).

1	2	3	4	5	6	7	8	9	10	dias(n) / entrenamientos consecutivos
0	0	0	0	0	0	0	0	0	0	**10**
0	0	0	0	0	0	0	0	0	0	**9**
0	0	0	0	0	0	0	0	0	0	**8**
0	0	0	0	0	0	0	0	0	0	**7**
0	0	0	0	0	0	0	0	0	0	**6**
0	0	0	0	0	0	0	0	0	0	**5**
0	0	0	0	0	0	0	0	0	0	**4**
0	0	0	0	0	0	0	0	0	0	**3**
0	0	0	0	0	0	0	0	0	0	**2**
0	**38**	0	0	0	0	0	0	0	0	**1**
36	2	0	0	0	0	0	0	0	0	**0**
36	**38**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	*Máximo alcanzado para el día i*

Si continuamos con esta línea de pensamientos llenamos la matriz obteniendo este resultado:

1	2	3	4	5	6	7	8	9	10	dias(n) / entrenamientos consecutivos
0	0	0	0	0	0	0	0	0	0	**10**
0	0	0	0	0	0	0	0	0	247	**9**
0	0	0	0	0	0	0	0	237	273	**8**
0	0	0	0	0	0	0	224	260	326	**7**
0	0	0	0	0	0	207	243	309	302	**6**
0	0	0	0	0	184	220	286	279	**380**	**5**
0	0	0	0	146	182	248	246	**342**	364	**4**
0	0	0	106	123	208	206	**309**	324	365	**3**
0	0	87	82	167	165	**268**	291	324	356	**2**
0	38	63	118	116	**219**	242	291	315	342	**1**
36	2	**99**	57	**158**	181	230	282	301	350	**0**
36	**38**	**99**	**118**	**167**	**219**	**268**	**309**	**342**	**380**	*Máximo alcanzado para el día i*

Las celdas marcadas en verde corresponden a los días entrenados para el problema de tamaño $n = 10$. Se puede observar que para cada día i (nuestros subproblemas) se obtiene el máximo obtenido. Es importante notar que por ejemplo, para $n = 5$ si bien el óptimo se obtiene con ganancia total de 167, no se incluye en la solución final. Esto nos lleva a no poder descartar las demás alternativas, al menos, para el día anterior al que estamos analizando.

Para definir la ecuación de recurrencia observamos el patrón utilizado para completar la matriz. En primer lugar, vamos a llamar G a una función que dado un n correspondiente a la cantidad de días y un j correspondiente a la cantidad de entrenamientos consecutivos, nos determine la ganancia obtenida, y luego OPT a aquella que nos determine la máxima posible:

$$G(n, j) = G(n - 1, j - 1) + \text{mín}(s_{j+1}, e_n)$$

Esta expresión es correcta pero no considera los casos "especiales", asi que vamos a extenderla:

- Si $n = 0$ no tenemos días para analizar y la ganancia obtenida es siempre 0. Esto también implica que $e_0 = 0$:
$$G(0, j) = 0$$
- Si $j = 0$ descansamos el día previo, por lo que debemos ir a buscar el óptimo del dia $n - 2$ y sumarle la ganancia obtenida con el máximo de energía:
$G(n, 0) = OPT(n - 2) + \text{mín}(s_1, e_n)$

$$\therefore G = \begin{cases} G(0, j) & = 0 \\ G(n, 0) & = OPT(n - 2) + \text{mín}(s_1, e_n) \\ G(n, j) & = G(n - 1, j - 1) + \text{mín}(s_{j+1}, e_n) \end{cases}$$

Con esta función podemos llenar cada casillero de la matriz. Ahora, como queremos obtener la ganancia máxima posible para n días, simplemente debemos considerar el mayor valor obtenido:

$$OPT(n) = \begin{cases} \text{máx}_{0 \leq j \leq n-1}\{G(n,j)\}, n > 0 \\ 0, n \leq 0 \end{cases}$$

3.3. Implementación del algoritmo

Decidimos trabajar con el lenguaje de programación Python dado que es muy amigable con el manejo de arrays y gráficos, siendo fácil de entender. Armamos una notebook jupyter para documentar mejor el código, que está organizado de la siguiente manera:

1. La sección inicial donde definimos
   - Función para obtener las ganancias y energías de los archivos.
   - Sección con los paths a utilizar
   - Función que obtiene los resultados esperados, devolviendo un array con el nombre del archivo, la ganancia máxima y el plan de entrenamiento que representa dicha ganancia.
2. La segunda sección donde se encuentran
   - La función que representa al algoritmo propuesto: `ganancia_maxima_posible(e_i, s_i)`
   - La función que reconstruye la solución: `reconstruir_solucion(ganancias, ganancia_maxima)`
   - La función que obtiene la máxima ganancia posible para el día según los días entrenados: `ganancia_dia_actual(e_i, s_i, dia, dias_entrenados)`
3. La tercer sección donde se realizan las pruebas correspondientes de nuestro algoritmo tanto para los sets propuetos por la catedra como los propios.
4. La cuarta y última sección que contiene el código del grafico hecho.

Nuestro algoritmo consiste en

1. Genenar una matiz inicial de $n * n$: `ganancias_obtenidas[dias][dias_entrenados]`

```
    cantidad_dias = len(e_i) # n
    ganancias_obtenidas = [[0] * cantidad_dias for _ in range(cantidad_dias)]
```

2. Obtener la ganancia para el primer día (caso base) e inicializar la cantidad de días previos que pude haber entrenado a 2

```
    dias_previos_entrenados = 2
    ganancias_obtenidas[0][0] = ganancia_dia_actual(e_i, s_i, 0, 0)
```

3. Iterar desde el día 1 hasta el n, completando la matriz con la ganancia obtenida de haber entrenado $0 \leq j \leq n$ días seguidos anteriormente

```
  for dia in range(1, cantidad_dias):
    ganancias_obtenidas[dia][0] = ganancia_dia_actual(e_i, s_i, dia, 0)
    ganancias_obtenidas[dia][0] += max(ganancias_obtenidas[dia-2] if dia > 1
    else [0])
    for dia_entrenado in range(1, dias_previos_entrenados):
      ganancias_obtenidas[dia][dia_entrenado] = ganancia_dia_actual(e_i, s_i,
    dia, dia_entrenado) + ganancias_obtenidas[dia-1][dia_entrenado-1]
    dias_previos_entrenados += 1
```

   La primera parte calcula la ganancia habiendo descansado el día anterior. En el for que le sigue, lo hace para cantidad de días entrenados consecutivos de 1 a dia-1.

4. Retornar la matriz para luego poder reconstruir la solución y el máximo obtenido en el último día, el cual representa la ganancia máxima posible.

```
    return ganancias_obtenidas, max(ganancias_obtenidas[cantidad_dias-1])
```

Para reconstruir la solución tenemos el siguiente algoritmo:

```
def reconstruir_solucion(ganancias, ganancia_maxima):
  n = len(ganancias) - 1
  solucion = []
  # recorrer la matriz bajando siempre por la diagonal hasta llegar
  # a la fila 0, luego ir al maximo de la columna-2
  dia = n - 1 # columnas
  dias_entrenados = ganancias[n].index(ganancia_maxima) - 1 # filas
  solucion.append('Entreno')
  while dia >= 0:
    while dias_entrenados >= 0 and dia >= 0:
      solucion.append('Entreno')
      dias_entrenados -= 1
      dia -= 1
    if dia >= 0:
      solucion.append('Descanso')
      # ir al maximo de la columna anterior
      dia -= 1
      dias_entrenados = ganancias[dia].index(max(ganancias[dia]))
  return solucion[::-1] # invertir la solucion
```

Es fácil de ver la lógica de esta reconstrucción en la imagen de la matriz de la sección 3.2. Necesitamos partir del máximo para el n que queramos reconstruír. Desde ahí, tenemos que descender de a un escalón y dado que la altura se corresponde con la cantidad de días entrenados previos, al llegar a la fila 0 significa que el día que le sigue a ese (de izquierda a derecha) descansamos. ¿Cómo continuamos al llegar al piso? Debemos saltar al máximo de la columna actual - 2. Dado que si descansamos, nos quedamos con el óptimo del día anterior. Luego, debemos continuar bajando escalones hasta finalizar los días.

Por último, invertimos el array para que nos quede en el orden esperado, es decir, acorde a n creciente.

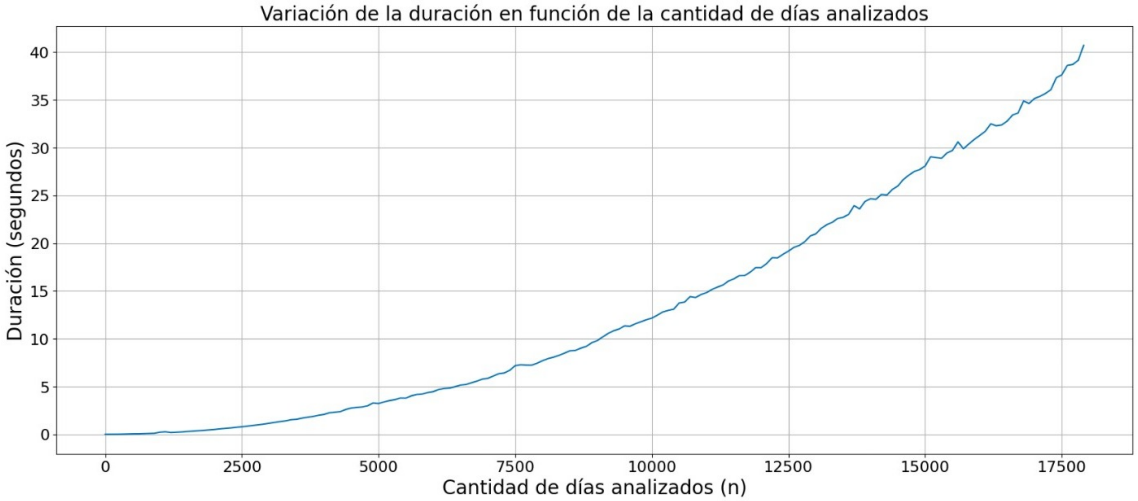
4. Complejidad del algoritmo

Al analizar la función que obtiene las ganancias máximas, observamos que se crea una matriz de ganancias de tamaño n * n, por lo que es notorio que la complejidad espacial es de $O(n^2)$. Luego se utilizan 2 bucles for para rellenar la matriz. El primer ciclo se ejecutará la cantidad de días brindados (n), mientras que el ciclo anidado se ejecuta la cantidad de días entrenados, siendo su máximo posible valor, la cantidad de días. Por lo tanto, la complejidad temporal en esta parte es $O(n^2)$. En total, la complejidad algoritmica es $O(n^2)$, y la complejidad espacial es $O(n^2)$ también, debido a la matriz.

Luego, si enfatizamos sobre nuestra reconstrucción de la solución, es notorio que la complejidad temporal es $O(n)$, ya que se recorren las n columnas de una matriz, que en nuestro caso son los días, y en cada columa se toma un valor de la misma y se elige una fila, que es igual a la cantidad de días entrenados. Finalmente se invierte el arreglo, siendo esta una operación $O(n)$ también. Respecto al espacio, nuestra solución crea una lista de tamaño n (cantidad de días).

5. Medición

Como se detalló anteriormente, nuestro problema crece a medida que hay más días en juego. Por lo que es de esperarse que este creciemiento se vea reflejado en el tiempo que tarda nuestro algoritmo en encontrar la solución óptima. Basándonos en esta hipótesis realizamos un gráfico que muesta el tiempo que tarda nuestro algoritmo en encontrar la solución a medida que crecen la cantidad de días. En cada iteración se sumo de a 100 días.



Al observar el gráfico, notamos un crecimiento parabólico a medida que aumenta la cantidad de días analizados, lo cual respalda nuestro análisis respecto de la complejidad algorítmica de la solución, ya que como comentamos anteriormente, esta misma es cuadrática. Lo mismo sucede espacialmente, al tener más días de entrenamiento, requerimos de más memoria para almacenar los resultados.

Algo interesante a destacar es que, no importa el set de datos con el cual se cuente, siempre nuestro algoritmo se comportará (en terminos de complejidad temporal y espacial) de igual forma. Con esto nos referimos a que si estamos frente a un set de datos donde la solución con mayor ganancia cambia con cada día o frente a uno donde todos los días se entrena o frente a cualquier set posible, en cada iteración se deben calcular todos los posibles caminos por los cuales pudimos haber llegado.

6. Conclusiones

Este trabajo práctico fue muy desafiante. Nos llevó mucho tiempo de análisis y de prueba y error. Incorporar la forma de pensamiento de Programación Dinámica es sin dudas difícil. Afortunadamente, pudimos llegar a una solución que hace uso de Memoization para evitar recálculos. Nos hubiera gustado hallar una que no necesitara tantos datos parciales para encontrar el resultado pero lo intentamos y no llegamos a nada correcto.

Al momento de escribir el código del algoritmo, si bien es sencillo, se nos iban ocurriendo algunas optimizaciones espaciales. Sin embargo, optamos por no aplicarlas porque no nos permitirían reconstruír la solución. Por otro lado, graficar para las mediciones también fue tedioso, dado que al aumentar la cantidad de datos, se hacia muy lento. Por eso optamos por generar muestras de a intervalos crecientes de 100 días.

Como comentario final nos gustaría decir que fue un trabajo interesante en el que tuvimos que cambiar nuestra forma de pensar para adaptarnos al problema en cuestión y nos gustó esa experiencia.